# Forms in React

## Handling Form Inputs in React

Forms in React are primarily controlled via the component's state. This means that React will manage the value of each input field, and changes to those fields will trigger state updates. We’ll go through how to capture user input, update the state, and render the input dynamically.

### Basic Form Example

In this example, we’ll create a simple form that allows the user to input their name and email.

import React, { useState } from 'react';

function SimpleForm() {

const [name, setName] = useState('');

const [email, setEmail] = useState('');

const handleNameChange = (event) => {

setName(event.target.value);

};

const handleEmailChange = (event) => {

setEmail(event.target.value);

};

const handleSubmit = (event) => {

event.preventDefault();

alert(`Name: ${name}, Email: ${email}`);

};

return (

<div>

<h1>React Form</h1>

<form onSubmit={handleSubmit}>

<div>

<label>Name: </label>

<input

type="text"

value={name}

onChange={handleNameChange}

/>

</div>

<div>

<label>Email: </label>

<input

type="email"

value={email}

onChange={handleEmailChange}

/>

</div>

<button type="submit">Submit</button>

</form>

</div>

);

}

export default SimpleForm;

### Explanation:

* We are using the useState hook to manage the form fields' values (name and email).
* Each input field is controlled, meaning its value is derived from React state. The onChange event handler updates the corresponding state when the user types in the input.
* The handleSubmit function prevents the form's default behavior (which would reload the page) and shows an alert with the values of name and email.

## Handling Multiple Form Fields

In real-world applications, forms often contain many fields. In React, you can manage multiple form fields with a single state object. This is useful for managing more complex forms and keeping your state logic clean and organized.

### Example: Handling Multiple Form Fields

import React, { useState } from 'react';

function MultiFieldForm() {

const [formData, setFormData] = useState({

name: '',

email: '',

password: ''

});

const handleChange = (event) => {

const { name, value } = event.target;

setFormData({

...formData,

[name]: value

});

};

const handleSubmit = (event) => {

event.preventDefault();

alert(`Name: ${formData.name}, Email: ${formData.email}, Password: ${formData.password}`);

};

return (

<div>

<h1>Multiple Form Fields</h1>

<form onSubmit={handleSubmit}>

<div>

<label>Name: </label>

<input

type="text"

name="name"

value={formData.name}

onChange={handleChange}

/>

</div>

<div>

<label>Email: </label>

<input

type="email"

name="email"

value={formData.email}

onChange={handleChange}

/>

</div>

<div>

<label>Password: </label>

<input

type="password"

name="password"

value={formData.password}

onChange={handleChange}

/>

</div>

<button type="submit">Submit</button>

</form>

</div>

);

}

export default MultiFieldForm;

### Explanation:

* Instead of having separate state variables for each form field, we use a single state object (formData) to manage all the fields.
* Each input field has a name attribute that matches the corresponding key in the formData object. When the user types in an input field, the handleChange function is triggered and updates the appropriate value in the state.
* This allows for easy management of multiple form fields, especially in larger forms.

## Example Project: Building a User Registration Form

Let's combine everything we’ve learned into a **User Registration Form** where users can enter their name, email, password, and confirm the password.

### Step-by-Step Code Creation:

1. **Create the React App**: If you haven't already done so, create a new React app:

npx create-react-app user-registration-form

cd user-registration-form

1. **Modify App.js**: Open src/App.js and update it with a user registration form:

import React, { useState } from 'react';

function RegistrationForm() {

const [formData, setFormData] = useState({

name: '',

email: '',

password: '',

confirmPassword: ''

});

const handleChange = (event) => {

const { name, value } = event.target;

setFormData({

...formData,

[name]: value

});

};

const handleSubmit = (event) => {

event.preventDefault();

if (formData.password !== formData.confirmPassword) {

alert("Passwords don't match");

return;

}

alert(`Registration Successful!\nName: ${formData.name}\nEmail: ${formData.email}`);

};

return (

<div>

<h1>User Registration</h1>

<form onSubmit={handleSubmit}>

<div>

<label>Name: </label>

<input

type="text"

name="name"

value={formData.name}

onChange={handleChange}

/>

</div>

<div>

<label>Email: </label>

<input

type="email"

name="email"

value={formData.email}

onChange={handleChange}

/>

</div>

<div>

<label>Password: </label>

<input

type="password"

name="password"

value={formData.password}

onChange={handleChange}

/>

</div>

<div>

<label>Confirm Password: </label>

<input

type="password"

name="confirmPassword"

value={formData.confirmPassword}

onChange={handleChange}

/>

</div>

<button type="submit">Register</button>

</form>

</div>

);

}

export default RegistrationForm;

### Explanation:

* This form collects the user's name, email, password, and confirms the password.
* We use a single formData state object to store all the field values.
* The handleChange function updates the corresponding field in the formData object whenever the user types.
* The form validates whether the password and confirm password fields match before submitting.

1. **Run the App**:

npm start